SERVLETS DEBUGGING

It is always difficult to testing/debugging a servlets. Servlets tend to involve a large amount of client/server interaction, making errors likely but hard to reproduce.

Here are a few hints and suggestions that may aid you in your debugging.

System.out.println()

System.out.println() is easy to use as a marker to test whether a certain piece of code is being executed or not. We can print out variable values as well. Additionally:

- Since the System object is part of the core Java objects, it can be used everywhere without the need to install any extra classes. This includes Servlets, JSP, RMI, EJB's, ordinary Beans and classes, and standalone applications.
- Compared to stopping at breakpoints, writing to System.out doesn't interfere much with the normal execution flow of the application, which makes it very valuable when timing is crucial.

Following is the syntax to use System.out.println():

```
System.out.println("Debugging message");
```

All the messages generated by above syntax would be logged in web server log file.

Message Logging:

It is always great idea to use proper logging method to log all the debug, warning and error messages using a standard logging method. I use log4J to log all the messages.

The Servlet API also provides a simple way of outputting information by using the log() method as follows:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ContextLog extends HttpServlet {
  public void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
         java.io.IOException {

      String par = request.getParameter("par1");
      //Call the two ServletContext.log methods
      ServletContext context = getServletContext( );

      if (par == null || par.equals(""))
      //log version with Throwable parameter
      context.log("No message received:",
          new IllegalStateException("Missing parameter"));
      else
          context.log("Here is the visitor's message: " + par);

      response.setContentType("text/html");
      java.io.PrintWriter out = response.getWriter( );
      String title = "Context Log";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
```

```
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<h2 align=\"center\">Messages sent</h2>\n" +
        "</body></html>");
    } //doGet
}
```

The ServletContext logs its text messages to the servlet container's log file. With Tomcat these logs are found in <Tomcat-installation-directory>/logs.

The log files do give an indication of new emerging bugs or the frequency of problems. For that reason it's good to use the log() function in the catch clause of exceptions which should normally not occur.

Using JDB Debugger:

You can debug servlets with the same jdb commands you use to debug an applet or an application.

To debug a servlet, we can debug sun.servlet.http.HttpServer, then watch as HttpServer executes servlets in response to HTTP requests we make from a browser. This is very similar to how applets are debugged. The difference is that with applets, the actual program being debugged is sun.applet.AppletViewer.

Most debuggers hide this detail by automatically knowing how to debug applets. Until they do the same for servlets, you have to help your debugger by doing the following:

- Set your debugger's classpath so that it can find sun.servlet.http.Http-Server and associated classes.
- Set your debugger's classpath so that it can also find your servlets and support classes, typically server_root/servlets and server_root/classes.

You normally wouldn't want server_root/servlets in your classpath because it disables servlet reloading. This inclusion, however, is useful for debugging. It allows your debugger to set breakpoints in a servlet before the custom servlet loader in HttpServer loads the servlet.

Once you have set the proper classpath, start debugging sun.servlet.http.HttpServer. You can set breakpoints in whatever servlet you're interested in debugging, then use a web browser to make a request to the HttpServer for the given servlet (http://localhost:8080/servlet/ServletToDebug). You should see execution stop at your breakpoints.

Using Comments:

Comments in your code can help the debugging process in various ways. Comments can be used in lots of other ways in the debugging process.

The Servlet uses Java comments and single line (// ...) and multiple line (/* ... */) comments can be used to temporarily remove parts of your Java code. If the bug disappears, take a closer look at the code you just commented and find out the problem.

Client and Server Headers:

Sometimes when a servlet doesn't behave as expected, it's useful to look at the raw HTTP request and response. If you're familiar with the structure of HTTP, you can read the request and response and see exactly what exactly is going with those headers.

Important Debugging Tips:

Here is a list of some more debugging tips on servlet debugging:

- Be aware that server_root/classes doesn't reload and that server_root/servlets probably does.

- Ask a browser to show the raw content of the page it is displaying. This can help identify formatting problems. It's usually an option under the View menu.
- Make sure the browser isn't caching a previous request's output by forcing a full reload of the page. With Netscape Navigator, use Shift-Reload; with Internet Explorer use Shift-Refresh.
- Verify that your servlet's init() method takes a ServletConfig parameter and calls super.init(config) right away.